Seventh Workshop on Non-Classical Models of Automata and Applications (NCMA 2015)

Short Papers

Preface

The *Seventh Workshop on Non-Classical Models of Automata and Applications (NCMA 2015)* has been organized to bring together researchers who work on various aspects of non-classical and classical models of automata, providing an excellent opportunity to develop and discuss novel ideas. Numerous models of automata, both classical and non-classical, are natural objects of theoretical computer science. They are studied from different points of view in various areas, both as theoretical concepts and as formal models for applications. The purpose of the NCMA workshop series is to promote a deeper and interdisciplinary coverage of this particular area and in this way foster new insights and substantial progress in computer science as a whole.

The first workshop on Non-Classical Models of Automata and Applications, NCMA 2009, was held in Wrocław, Poland, in 2009 as a satellite event of the 17th International Symposium on Fundamentals of Computation Theory (FCT 2009). It was sponsored by the AutoMathA project of the European Science Foundation (ESF). The second workshop, NCMA 2010, was held in Jena, Germany, as an associated workshop of the 11th International Conference on Membrane Computing (CMC 11), and the third workshop, NCMA 2011, was organized at the Università degli Studi di Milano, Milan, Italy, in close proximity to the 15th Conference on Developments in Language Theory (DLT 2011). In contrast to the previous workshops, the next NCMA workshops were organized as stand-alone events; NCMA 2012 was held in Fribourg, Switzerland, in August 2012, NCMA 2013 took place in Umeå, Sweden, in August 2013, and NCMA 2014 was held in Kassel, Germany, in July 2014. Now the Seventh Workshop on Non-Classical Models of Automata and Applications (NCMA 2015) takes place in Porto, Portugal, on August 31st and September 1st, 2015. We expect it to be again a scientifically valuable event with interesting presentations, exciting results, and stimulating discussions. We hope that the friendly atmosphere and the nice surroundings will also help to make this workshop a worthwhile event that will lead to new insights and possibly new cooperations.

At NCMA 2015 there are three invited presentations given by

- Patricia Bouyer-Decitre (LSV, CNRS and ENS de Cachan, France)
  On the Optimal Reachability Problem in Weighted Timed Automata and Games,
- Sylvain Lombardy (LaBRI, Bordeaux, France)
  On Two-Way Weighted Automata, and
- Emanuele Rodaro (CMUP, University of Porto, Portugal) on
  New Reformulations of Cerny's Conjecture and Related Problems.

We thank them for accepting our invitation and for presenting their recent results.

For NCMA 2015, we received submissions by a total of 22 authors from 8 different countries. From these submissions, on the basis of three referees' reports each, the Program Committee selected 9 contributions for presentation at NCMA and for inclusion in the workshop proceedings, which appeared as volume 318 in the series books@ocg.at of the Austrian Computer Society.

In addition to the invited presentations and the regular contributions, NCMA 2015 also features four short presentations to emphasize its workshop character, each of them also having been evaluated by at least two referees. The extended abstracts of these short presentations are contained in this booklet.

We want to thank the members of the Organizing Committee for their efforts in the preparation of the workshop (all from the University of Porto): Ivone Amorim, Sabine Broda, António Machiavelo, Eva Maia, Nelma Moreira, and Rogério Reis.

We are grateful to the Centre of Mathematics of University of Porto and the Computer Science Department of Science Faculty of University of Porto for the support in the local organization of NCMA 2015, to the University of Porto for financial support, and to the Institute of Computer Languages of the Vienna University of Technology for covering the printing costs of the proceedings and of this booklet of short papers.

August 2015

Rudolf Freund, Vienna
Markus Holzer, Giessen
Nelma Moreira, Porto
Rogério Reis, Porto

Table of Contents

Short Papers

ON LEFT-MOST HAIRPIN FINITE AUTOMATA

Henning Bordihn[(A)] Erzsébet Csuhaj-Varjú[(B)]
György Vaszil[(C)]

[(A)]Institute of Computer Science, University of Potsdam,
August-Bebel-Straße 89, 14482 Potsdam, Germany
henning@cs.uni-potsdam.de

[(B)]Eötvös Loránd University, Faculty of Informatics,
Pázmány Péter sétány 1/C, H-1117 Budapest, Hungary
csuhaj@inf.elte.hu

[(C)]Department of Computer Science, University of Debrecen,
H-4010 Debrecen, P.O. Box 12, Hungary
vaszil.gyorgy@inf.unideb.hu

Abstract

We show that left-most hairpin automata can accept only semi-linear languages and that non-regular languages can be accepted if at least four different letters of the input alphabet can be used as pointer symbols. We also examine the case when left-most hairpin automata are reversible, we show that in this case only regular languages can be accepted.

1. Introduction

Deterministic finite state automaton is one of the most traditional and best investigated devices of theoretical computer science. Since its introduction several variants and extensions of this model have been considered. One example is the notion of *extended finite state automata*, where the machine may perform an additional operation on the non-read part of the input word, see [1, 2, 3, 4, 5]

In [4] and [5], the hairpin operation was used, where designated subwords of the remaining input word can be inverted. The hairpin operation is inspired by the biological process of gene assembly performed during the replication of single-cell organisms, see [6, 7]. Although the models proposed in [6] and [7] are different, both are based on simple operations on the DNA molecule guided by pointers. In the latter model the used operations are inspired by the way in which a DNA molecule can fold. One of these operations is the hairpin loop with inverted

pointers that reverses a substring between a pointer p and the reversal of p. The hairpin inverse of a word w in Σ^+ is defined as

$$hi(w) = \{\, xay^R az \mid w = xayaz \text{ and } x, y, z \in \Sigma^*,\ a \in \Sigma \,\}.$$

where a serves as pointer (by using appropriate morphisms it is enough to consider pointers of length 1 only.)

In hairpin finite automata, the hairpin operation can be applied to a prefix of the remaining (non-read) input word whenever a pointer is read. As to the second pointer involved in the operation, three different modes of hairpin inversion are distinguished, namely left-most hairpin (the second pointer is as close as possible), general hairpin (no restriction on the position of the second pointer), and right-most hairpin (the second pointer is as far away as possible). In [4, 5] the computational capacity of right-most and general hairpin finite automata is investigated, whereas the computational power of left-most hairpin finite automata is listed as an open question. In the present paper we provide some initial results in this direction.

2. Preliminaries

For a set S, we denote its powerset by 2^S. Set inclusion is denoted by $\subseteq$, and strict set inclusion by $\subset$. For a set M, $\mathrm{card}(M)$ is the cardinality of M.

An alphabet V is a finite nonempty set of symbols. For an alphabet V, we use the following notations: the set of all words over V is denoted by V^*, the empty word is denoted by λ, the reversal of a word w by w^R, and for the length of w we write $|w|$. For the number of occurrences of a symbol a in w we use the notation $|w|_a$. A language L is a subset of V^*.

In the following we recall the notions of extended and hairpin finite state automata, as given in [3] and subsequent papers.

Definition 2.1 *A* (nondeterministic) extended finite state automaton *is a* 6*-tuple* $A = (Q, \Sigma, \delta, \Delta, q_0, F)$*, where* Q *is a finite set of states,* Σ *is the input alphabet,* δ *and* Δ *are mappings from* $Q \times (\Sigma \cup \{\lambda\})$ *to* 2^Q*, where* δ *is called the transition function,* $q_0 \in Q$ *is the initial state, and* $F \subseteq Q$ *is the set of accepting states. Furthermore,* A *is said to be* λ-free*, if both* δ *and* Δ *are restricted to* $Q \times \Sigma$*.*

A *configuration* of A is a tuple (q, w), where $q \in Q$ is the current state, and $w \in \Sigma^*$ is the still unread part of the input. If a is in $\Sigma \cup \{\lambda\}$ and w in Σ^*, then we write $(q, aw) \vdash_A (p, w)$, if p is in $\delta(q, a)$. Those transitions will be referred to as ordinary transitions.

A hairpin operation is performed by applying the mapping Δ. For $a \in \Sigma$ and $v, w \in \Sigma^*$, a *left-most hairpin transition* is defined by $(q, avaw) \vdash_A (p, av^Raw)$, for p in $\Delta(q, a)$ or $\Delta(q, \lambda)$, and $v \in (\Sigma \setminus \{a\})^*$. The corresponding transitions will be referred to as non-ordinary transitions.

An extended finite state automaton $A = (Q, \Sigma, \delta, \Delta, q_0, F)$ with left-most hairpin transitions is called a *left-most hairpin finite automaton (lh-FA)*,

Whenever there is a choice between an ordinary transition or a hairpin operation, the automaton non-deterministically chooses the next move. As usual, the reflexive transitive closure of $\vdash_A$ is denoted by $\vdash_A^*$. The subscript A will be dropped from $\vdash_A$ and $\vdash_A^*$ whenever the meaning remains clear.

The language accepted by A is the set $T(A) = \{ w \mid (q_0, w) \vdash_A^* (q, \lambda), q \in F \}$.

For examples of hairpin finite automata and their languages as well as some fundamental results, the reader is referred to [5].

3. On the Computational Power of Left-Most Hairpin Finite Automata

The first result applies not only to languages accepted by left-most hairpin finite automata, but is equally valid for any family of languages that can be accepted by some extended finite automaton, deterministic or non-deterministic, as long as the operation performed in non-ordinary transitions does not consume or replace input symbols, such as hairpin-automata of any type, input-reversal or input-revolving finite automata.

Theorem 3.1 *Every language that can be accepted by a hairpin finite automaton is semi-linear.*

Proof. Let L be a language accepted by a left-most hairpin finite automaton. We show that L is letter-equivalent to a regular language. Let $A = (Q, \Sigma, \delta, \Delta, q_0, F)$ be the (λ-free) hairpin finite automaton accepting L. Every computation of A comprises of ordinary and non-ordinary transitions. In non-ordinary transitions, where a hairpin operation is applied, the order of the symbols of designated subwords is changed, leading to some permutation of the symbols of the (remaining part of the) input word. In ordinary transitions, input symbols are consumed as in ordinary finite state automata.

Consider any accepting computation of A, say on input word w. The permutation of w such that A can consume it only by the ordinary transitions that are performed in the accepting computation on w, can be processed by a non-deterministic finite state automaton, if appropriate spontaneous state changes are allowed on pointer symbols. More precisely, let $A' = (Q \cup \bar{Q}, \Sigma, \delta', q_0, F)$ be the non-deterministic finite state automaton allowing λ-transitions with $\bar{Q} = \{ \bar{q} \mid q \in Q \}$, $Q \cap \bar{Q} = \emptyset$, and $\delta' = \delta \cup \delta''$, where δ'' is defined as follows: If $p \in \Delta(q, a)$, then

$$\bar{p} \in \delta''(q, \lambda) \text{ and } \delta''(\bar{p}, a) = \delta(p, a).$$

Now, a word is accepted by A' if and only if it is a permutation of some word from $T(A)$. Hence, $T(A)$ is letter-equivalent to the regular language accepted by A'. □

The complexity of the structure of the words which can be accepted by left-most hairpin finite automata depends on the number of input symbols that can be treated as pointers.

Proposition 3.2 *There is a non-regular language that can be accepted by a left-most hairpin finite automaton A with four pointers.*

Proof. Consider $L = \{\, cad\#(cadacada)^n cad\$cb\#(db\$db\#cb\$cb\#)^n db\$ \mid n \geq 0 \,\}$. This language is not regular as the image under the morphism h defined by $h(a) = a$, $h(b) = b$ and $h(x) = \lambda$ for $x \in \{c, d, \#, \$\}$, namely $h(L) = \{\, a^{4n+2}b^{4n+2} \mid n \geq 0 \,\}$ is not regular, and the family of regular languages is closed under homomorphisms.

We show that L is accepted by the following hairpin finite automaton $A = (Q, \Sigma, \delta, \Delta, q_0, F)$ working in the left-most mode. We set $F = \{q_a\}$ and $\delta(q_0, c) = \{q_l, q_f\}$.

If A enters q_f in the first step of a computation, then A starts a sequence of ordinary transitions reading the input $cad\#cad\$cb\#db\$$, that is, the shortest word of the language, where $n = 0$. This sequence of ordinary transitions leads A into its only accepting state q_a, and this is the only way how q_a can be reached from q_f. Thus, the word with $n = 0$ is accepted.

If A enters q_l upon reading a c in q_0, then it starts another computation which aims to reduce an input string of the form

$$cad\#(cadacada)^{n+1}cad\$cb\#(db\$db\#cb\$cb\#)^{n+1}db\$ \tag{1}$$

to the string

$$cad\#(cadacada)^{n}cad\$cb\#(db\$db\#cb\$cb\#)^{n}db\$, \tag{2}$$

thus, after reading the first 20 symbols of the input of the form (1), it will enter the state q_0 again, and have the string (2) on its input tape.

This reduction is performed by a deterministic sub-computation stepping through a sequence of states that is disjoint from the sequence starting from q_f, and in which no state is used twice. Let $Q = \{q_0, q_a, q_l, q_f\} \cup \{q_i \mid 1 \leq i \leq 27\}$, and let

$$\begin{aligned}
\delta(q_l, c) &= q_1, \quad \delta(q_1, a) = q_2, \quad \delta(q_2, d) = q_3,\\
\Delta(q_3, \#) &= q_4,\\
\delta(q_4, \#) &= q_5, \quad \delta(q_5, b) = q_6,\\
\Delta(q_6, c) &= q_7,\\
\delta(q_7, c) &= q_8, \quad \delta(q_8, a) = q_9, \quad \delta(q_9, d) = q_{10},\\
\Delta(q_{10}, \$) &= q_{11},\\
\delta(q_{11}, \$) &= q_{12}, \quad \delta(q_{12}, b) = q_{13},\\
\Delta(q_{13}, d) &= q_{14},\\
\delta(q_{14}, d) &= q_{15}, \quad \delta(q_{15}, a) = q_{16}, \quad \delta(q_{16}, c) = q_{17},\\
\Delta(q_{17}, \#) &= q_{18},\\
\delta(q_{18}, \#) &= q_{19}, \quad \delta(q_{19}, b) = q_{20},\\
\Delta(q_{20}, d) &= q_{21},\\
\delta(q_{21}, d) &= q_{22}, \quad \delta(q_{22}, a) = q_{23}, \quad \delta(q_{23}, c) = q_{24},\\
\Delta(q_{24}, \$) &= q_{25},\\
\delta(q_{25}, \$) &= q_{26}, \quad \delta(q_{26}, b) = q_{27},\\
\Delta(q_{27}, c) &= q_0.
\end{aligned}$$

This reduction is performed according to the schema presented in Table 1.

This sub-computation leads the automaton back to q_0, such that it can either repeat this sub-computation or consume $cad\#cad\$cb\#db\$$ by entering q_f when reading the first letter c. As no

Remaining suffix of the input word	Transition(s)
$cad\#(cadacada)^{n+1}cad\$cb\#(db\$db\#cb\$cb\#)^{n+1}db\$$	3 ordinary transitions
$\#(cadacada)^{n+1}cad\$cb\#(db\$db\#cb\$cb\#)^{n+1}db\$$	hairpin operation on #
$\#bc\$dac(adacadac)^{n+1}\#(db\$db\#cb\$cb\#)^{n+1}db\$$	*is equal to ...*
$\#bc\$dacadac(adacadac)^{n}adac\#db\$db\#cb\$cb\#(db\$db\#cb\$cb\#)^{n}db\$$	2 ordinary transitions
$\boldsymbol{c}\$da\boldsymbol{c}adac(adacadac)^{n}adac\#db\$db\#cb\$cb\#(db\$db\#cb\$cb\#)^{n}db\$$	hairpin operation on c
$\boldsymbol{c}ad\$\boldsymbol{c}adac(adacadac)^{n}adac\#db\$db\#cb\$cb\#(db\$db\#cb\$cb\#)^{n}db\$$	3 ordinary transitions
$\boldsymbol{\$}cadac(adacadac)^{n}adac\#db\boldsymbol{\$}db\#cb\$cb\#(db\$db\#cb\$cb\#)^{n}db\$$	hairpin operation on $\$$
$\boldsymbol{\$}bd\#cada(cadacada)^{n}cadac\boldsymbol{\$}db\#cb\$cb\#(db\$db\#cb\$cb\#)^{n}db\$$	2 ordinary transitions
$\boldsymbol{d}\#ca\boldsymbol{d}a(cadacada)^{n}cadac\$db\#cb\$cb\#(db\$db\#cb\$cb\#)^{n}db\$$	hairpin operation on d
$\boldsymbol{d}ac\#\boldsymbol{d}a(cadacada)^{n}cadac\$db\#cb\$cb\#(db\$db\#cb\$cb\#)^{n}db\$$	3 ordinary transitions
$\#da(cadacada)^{n}cadac\$db\#cb\$cb\#(db\$db\#cb\$cb\#)^{n}db\$$	hairpin operation on #
$\#bd\$cadac(adacadac)^{n}ad\#cb\$cb\#(db\$db\#cb\$cb\#)^{n}db\$$	2 ordinary transitions
$\boldsymbol{d}\$ca\boldsymbol{d}ac(adacadac)^{n}ad\#cb\$cb\#(db\$db\#cb\$cb\#)^{n}db\$$	hairpin operation on d
$\boldsymbol{d}ac\$\boldsymbol{d}ac(adacadac)^{n}ad\#cb\$cb\#(db\$db\#cb\$cb\#)^{n}db\$$	3 ordinary transitions
$\boldsymbol{\$}dac(adacadac)^{n}ad\#cb\boldsymbol{\$}cb\#(db\$db\#cb\$cb\#)^{n}db\$$	hairpin operation on $\$$
$\boldsymbol{\$}bc\#da(cadacada)^{n}cad\boldsymbol{\$}cb\#(db\$db\#cb\$cb\#)^{n}db\$$	2 ordinary transitions
$\boldsymbol{c}\#da(\boldsymbol{c}adacada)^{n}cad\boldsymbol{\$}cb\#(db\$db\#cb\$cb\#)^{n}db\$$	hairpin operation on c
$\boldsymbol{c}ad\#(\boldsymbol{c}adacada)^{n}cad\boldsymbol{\$}cb\#(db\$db\#cb\$cb\#)^{n}db\$$	

Table 1: Sequence of remaining suffixes of the input word in a computation from q_l in the left column and the transitions to be performed on it in the right column of the same row.

further transitions are defined, the automaton A accepts the language L. □

It is an interesting open problem to consider automata with two or three symbols as pointers (languages accepted with one pointer are clearly regular). We have the following

Conjecture 3.3 *Left-most hairpin automata with two pointers accept regular languages.*

We close the paper with a statement on the regularity of languages acceptable by hairpin finite automata, but instead of restrictions on the number of pointers, we impose restrictions on the automata themselves, namely, we require them to be reversible. In what follows, let $P(A) = \{\, a \in \Sigma \mid \text{there is a state } q \text{ such that } \Delta(q, a) \neq \emptyset \,\}$.

Definition 3.4 *A left-most hairpin finite automaton $A = (Q, \Sigma, \delta, \Delta, q_0, F)$, is called* reversible*, if each configuration uniquely determines its preceding one, that is, there are no states $q, r \in Q$, $q \neq r$, such that $\delta(q, a) = \delta(r, a)$ for some $a \in \Sigma$, and moreover, for any $a \in P(A)$, if $\delta(q, a) = r$, then by observing r it can be determined whether a is found on the right-end of a substring of the input which is reversed by a hairpin inversion during this particular computation.*

Before turning the next result we define the set $R(A)$ of words that are read through by A during the acceptance process of every word u in $T(A)$.

More formally, let the word $w = a_1 \dots a_m$ be accepted by the hairpin automaton A through the transition sequence $(q_0, a_1 \dots a_m) \vdash_A^* (q, \lambda), q \in F$ and let $\delta(q_0, a_{i_1})$, $\delta(q_{i_1}, a_{i_1})$, $\delta(q_{i_2}, a_{i_2}), \dots,$ $\delta(q_{i_t}, a_{i_t})$ be the sequence of ordinary transitions used by A. Then the string $a_{i_1} a_{i_2} \dots a_{i_t}$ is an element of $R(A)$.

Theorem 3.5 *If L is accepted by a reversible left-most hairpin finite automaton A then L is regular.*

Proof. (**Sketch**) Consider the word $v \in R(A)$ corresponding to the input string $u \in T(A)$. Since A is reversible, its computation on u, producing v on the input tape, can be run "backwards". This means that an A' left-most hairpin automaton can be constructed, which when run on the v^R, the reverse of v, actually reads u^R the reverse of u from the input tape, thus, $u^R \in R(A')$ for any $u \in T(A)$. Since $R(A')$ is the set of words which are actually read through by A' during the acceptance process, the language $R(A')$ is regular, which means (as regular languages are closed under reversal) that the language $T(A)$ accepted by the reversible left-most hairpin finite automaton A is also regular. □

References

[1] S. BENSCH, H. BORDIHN, M. HOLZER, M. KUTRIB, Deterministic input-reversal and input-revolving finite automata. In: C. MARTÍN-VIDE, F. OTTO, H. FERNAU (eds.), *Language and Automata Theory and Applications (LATA 2008)*. Lecture Notes in Computer Science 5196, Springer-Verlag, 2008, 113–124.

[2] S. BENSCH, H. BORDIHN, M. HOLZER, M. KUTRIB, On input-revolving deterministic and nondeterministic finite automata. *Information and Computation* 207 (2009), 1140–1155.

[3] H. BORDIHN, M. HOLZER, M. KUTRIB, Revolving-Input Finite Automata. In: C. DE FELICE, A. RESTIVO (eds.), *Developments in Language Theory (DLT 2005)*. Lecture Notes in Computer Science 3572, Springer-Verlag, 2005, 168–179.

[4] H. BORDIHN, M. HOLZER, M. KUTRIB, Hairpin finite automata. In: T. HARJU, J. KARHUMÄKI, A. LEPISTÖ (eds.), *Developments in Language Theory (DLT 2007)*. Lecture Notes in Computer Science 4588, Springer-Verlag, 2007, 108–119.

[5] H. BORDIHN, M. HOLZER, M. KUTRIB, Hairpin finite automata. *Journal of Automata, Languages and Combinatorics* 16 (2011), 91–107.

[6] A. EHRENFEUCHT, D. PRESCOTT, G. ROZENBERG, Computational aspects of gene (un)scrambling in ciliates. In: L. LANDWEBER, E. WINFREE (eds.), *Evolution as Computation*. Springer-Verlag, 2002, 216–256.

[7] L. KARI, L. LANDWEBER, Computational power of gene rearrangement. In: E. WINFREE, D. GIFFORD (eds.), *DNA Based Computers V*. DIMACS 54, AMS Press, 2000, 207–216.

VALUE AUTOMATA WITH FILTERS*

Michaël Cadilhac[(A)] Andreas Krebs[(A)]
Nutan Limaye[(B)]

[(A)]WSI, Universität Tübingen
{cadilhac,krebs}@informatik.uni-tuebingen.de

[(B)]Indian Institute of Technology Bombay
nutan@cse.iitb.ac.in

Abstract
We propose to study value automata with filters, a natural generalization of regular cost automata to nondeterminism. Models such as weighted automata and Parikh automata appear naturally as specializations. Results on the expressiveness of this model offer a general understanding of the behavior of the models that arise as special cases. A landscape of such restrictions is drawn.

1. Introduction

Through their characteristic functions, formal languages are naturally seen as giving weights in $\{0, 1\}$ to words. The transparent correspondence between these two views is provided by the notion of *weighted automata*, a widely studied model firmly rooted in sound algebraic concepts (see [12, 6] for recent expositions). Weighted automata provide an elegant framework to capture some functions from words to a value set, and their frequent use in the modeling of the qualitative aspects of real-life systems [2, 5] is a witness of their richness. In its simpler form, a weighted automaton can be thought as an automaton where transitions bear integer weights; on reading a word, the weights are multiplied along the path, and the final values for all paths labeled by the word are summed. This model is thus in some sense "restricted" to update its single register by multiplying it by constants along the path, and summing its different possible values.

In an effort to develop a theory of cost functions with a wider spectrum of update mechanisms, Alur *et al.* [1] introduced *regular cost functions*, a highly parametrizable framework in which a *variant* of weighted automata functions arises as a particular case. In doing so, they also introduced a *deterministic* model of automata, *cost register automata* (CRA), and studied in which settings this model matches regular cost functions. Similarly to regular cost functions, CRA are defined with respect to an underlying algebraic structure and restrictions on the update

*Ongoing work.

[(B)]The work of this paper was done during a stay of the third author at the Universität Tübingen.

functions, but in this context, there exists an instance of these parameters that makes CRA precisely equivalent to weighted automata.

This correspondence between CRA and weighted automata is however not a straightforward matter of renaming: weighted automata are intrinsically nondeterministic machines, while CRA are defined as deterministic automata. With an appropriate generalization of CRA to nondeterminism, in which weighted automata would arise as an obvious special case, the aforementioned correspondence would thus express that this case is *determinizable*.

Hence, the starting point of the present research is to propose a nondeterministic generalization of CRA that encompasses the behavior of weighted automata in a natural fashion. By computing on several registers at once, CRA however provide more information than weighted automata, and simply collapsing these registers into a single value seems to be a loss. We thus propose to add a *filtering* step at the end of the computation: the vector of registers should lay in a prescribed set. This allows models such as reversal-bounded counter machines [9], Parikh automata [10], or finite automata over free groups [11] to naturally have a quantitative counterpart. The fact that the test is made *at the end* of the computation ensures that the models stay decidable

This framework is sufficiently well-behaved to allow for general results that echo those to be found within special cases. It also offers a unified view of the limits inherent to submodels, where questions such as the following can be asked: "which properties of the underlying algebraic, update, and filter parameters are required for closure under a certain operation to hold?"

2. Definitions and Preliminaries

A *monoid* is a set equipped with a binary associative operation having an identity element. For $(M, +)$ and $(N, \times)$ two monoids, a morphism $h\colon M \to N$ is a function such that $h(1) = 1$ and $h(ab) = h(a)h(b)$, for $a, b \in M$. A *semiring* is a set K equipped with two binary operations $+$ and $\times$ such that $(R, +)$ is a commutative monoid with identity 0, $(R, \times)$ is a monoid with identity 1 and absorbing element 0, and $\times$ distributes over $+$.

A *semiautomaton* A is a tuple (Q, Σ, δ), where Q is a set of states, Σ is an alphabet, and $\delta \subseteq Q \times \Sigma \times Q$ is a set of transitions. For a word w, we write $\mathsf{Paths}(A, w) \subseteq \delta^*$ for the set of paths on A labeled w. For a path π, we write $\mathsf{Orig}(\pi)$, resp. $\mathsf{Dest}(\pi)$, for the state in which the path starts, resp. ends. The semiautomaton is said to be *unambiguous* if there does not exist two different paths with same origin, destination, and label.

A *weighted automaton* W over a semiring $K = (K, +, \times)$ is a tuple (A, I, U, F) such that $A = (Q, \Sigma, \delta)$ is a semiautomaton, $U\colon \delta^* \to (K, \times)$ is a morphism, and $I, F\colon Q \to K$. For a word $w \in \Sigma^*$, the automaton computes the value $W(w)$ defined by:

$$W(w) = \sum_{\pi \in \mathsf{Paths}(A,w)} I(\mathsf{Orig}(\pi)) \times U(\pi) \times F(\mathsf{Dest}(\pi)) \ ,$$

where $\sum$ is the iterated $+$. The class of functions computed by weighted automata on K is denoted $\mathrm{WA}(K)$. Moreover, W is *unambiguous* if A is, and we denote $\mathrm{UnWA}(K)$ the class of

functions computed by unambiguous weighted automata on K—note that in such automata, the actual interpretation of $+$ is not needed.

3. Value Automata with Filters

Informally, this model differs from a weighted automaton on a semiring $(K, +, \times)$ in two aspects:

1. The weighted automaton updates *its only* register x with actions of the form $x \leftarrow x \times c$, where c is determined by the transition. In value automata, more than one register can appear, and the update expressions can be algebraic expressions.
2. If a word is recognized by n different paths, the values $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$ of x at the end of each path are *collapsed* using $+$. In contrast, value automata with filters apply a sieve on the different values of x, collapsing only those belonging to a prescribed set.

With generalization in mind, we wish to present a formalization that does not impose any *a priori* structure on the objects at hand, namely the underlying algebraic structure, the update functions, and the filter sets.

3.1. Algebraic Structures and Classes

Definition 3.1 *An* algebraic structure *K is composed of a base set (written also K) and multiple internal operations, one of which is a distinguished operation called the* collapse*. We assume that this operation is commutative, can take an unbounded number of arguments (associativity implying such a property), and acts as the identity when applied on a single element. This very general concept allows K to be a monoid, a semiring, etc.*

Let K be an algebraic structure and $f\colon K^m \to K^n$ be a function. We will always assume that the functions can be expressed using algebraic expressions, that is: each entry of $f(\vec{x})$ can be defined using an algebraic expression relying on the operations of K and the variables in $\vec{x}$. This corresponds to the idea that f should be an update function, and should not, for instance, make tests if K is not itself equipped with a test mechanism. Further, we say that f is *copyless* if the set of expressions used to define all the entries of $f(\vec{x})$ contains at most one occurrence of each component of $\vec{x}$. We say that f is *moveless* if for all i, the i-th component of $f(\vec{x})$ is not influenced by the values of x_j, $j \neq i$. Finally, we say that f is *resetless* if no component of $f(\vec{x})$ is constant when $\vec{x}$ varies.

Definition 3.2 *An* update class Upd *on K is a function that maps each pair $(m, n) \in \mathbb{N} \times \mathbb{N}$ to a set of functions from K^m to K^n. We add in superscript* cl*,* ml*, or* rl *when we consider the copyless, moveless, or resetless subsets of* Upd *(e.g.,* $\mathsf{Upd}^{\mathrm{cl,ml}}$*). In our presentation, update classes will serve as restrictions on the register updates.*

Examples 1

- When no restriction is imposed, that is, when updates are of the form of any algebraic expressions on K, we denote the update class $\top$.

- Let $+$ and $\times$ be two distinguished operations of K. We let Affine be the update class of affine functions, i.e., every $f \in \mathsf{Affine}(m,n)$ is defined using a matrix $K^{n\times m}$ and a vector $\vec{v} \in K^n$ by $f(\vec{x}) = M.\vec{x} + \vec{v}$. This class is succinctly written "$+, \times c$" in [1], as it corresponds to updates using additions and multiplications by constants.
- With K as previously, 0 and 1 respectively absorbing and neutral elements for $\times$, and $m = n$, restrain Affine by having M be a 0–1-matrix with exactly one 1 per row. We obtain the update class Trans, so denoted as updates are of the form $x_i \leftarrow x_j + c$. This class is written "$+c$" in [1].
- Similarly, when restraining Affine by having $\vec{v}$ be the null vector, and M have at most one nonzero entry per row, we obtain the update class Scale, corresponding to updates of the form $x_i \leftarrow x_j \times c$.

Definition 3.3 *A* filter class Filt *on* K *is a map* $d \in \mathbb{N} \mapsto 2^{K^d}$.

Examples 2
- The filter class $\top$ maps d to K^d. As a filter, its action is then void.
- Let $+$ be a distinguished operation of K and $<$ an order on K. The FO[$+$]-definable sets are those expressible as a first-order formula using $+$, $<$, and constants. More precisely, $C \subseteq K^d$ is FO[$+$]-definable if there is such a formula φ with d free variables such that $\vec{x} \in C$ iff $K \models \varphi(\vec{x})$. We write this filter class FO[$+$]. For $K = \mathbb{N}, \mathbb{Z}$ this corresponds to the *semilinear sets*, that is, finite unions of sets of the form $\vec{v} + K.\vec{v'} + K.\vec{v''} + \cdots$, where the $\vec{v}$'s are in K^d [8]. We write $\emptyset$[$+$] for the *quantifier-free* variant of this filter class, that is, sets corresponding to formulas with no quantifiers.

3.2. Value Automata with Filters

Definition 3.4 (Value Automaton with Filter) *Let* K *be an algebraic structure,* Upd *an update class on* K*, and* Filter *a filter class on* K*. Write* $\oplus$ *for the collapse operation of* K*.*

A Value Automaton with Filter (VAF) of dimension d *is a tuple* $\mathcal{A} = (A, I, U, C, F)$ *where:*
- $A = (Q, \Sigma, \delta)$ *is a semiautomaton,*
- $I \colon Q \to K^d$ *is a partial function called the* initialization*,*
- $U \colon \delta \to \mathsf{Upd}(d,d)$ *is the* update function*,*
- $C \subseteq K^d$ *is a subset in* $\mathsf{Filter}(d)$ *called the* filter*.*
- $F \colon Q \to \mathsf{Upd}(d,1)$ *is a partial function called the* finalization*.*

The VAF $\mathcal{A}$ *defines a* partial *function* $f \colon \Sigma^* \to K$ *as follows. Intuitively, for a path in* A *labeled* w*, the registers are initialized with* $I(q)$*, for* q *the first state of the path. They are then updated using the functions of* U*, filtered by* C*, and merged into a single value by* $F(q')$*, for* q' *the last state of the path. All such values for a label* w *are then collapsed using the collapse operation.*

Formally, write U_t *for* $U(t)$*, and* F_q *for* $F(q)$*. Define the valuation of a path* $\mathsf{val} \colon \delta^+ \to K^d$ *by* $\mathsf{val}(t) = I(\mathsf{Orig}(t))$*, for* $t \in \delta$*, and* $\mathsf{val}(\pi t) = U_t(\mathsf{val}(\pi))$*, with* $\pi \in \delta^*$*,* $t \in \delta$*. In particular, if a path starts in* q *and* $I(q)$ *is undefined, then the valuation of the path is undefined. Finally:*

$$f(w) = \bigoplus_{\substack{\pi \in \mathsf{Paths}(A,w) \\ \mathsf{val}(\pi) \in C}} F_{\mathsf{Dest}(\pi)}(\mathsf{val}(\pi)) \ ,$$

where we implicitly discard undefined values of $\mathsf{val}(\pi)$*, and the* $\oplus$ *of zero elements is undefined.*

The VAF is deterministic (DetVAF) if A is deterministic and I is defined on a single state. It is unambiguous (UnVAF) if A is. Finally, it is one-success (OneVAF) if $|\{\pi \in \mathsf{Paths}(A, w) \mid \mathsf{val}(\pi) \in C\}| \leq 1$*, that is, if the collapse is not needed.*

The class of such automata is written $\mathrm{VAF}(K, \mathsf{Upd}, \mathsf{Filter})$*, and similarly for DetVAF, UnVAF, and OneVAF. We identify these classes with the classes of functions they define. For a distinguished element* 0 *of* K*, the* 0-support *of a VAF is set of words it maps to* 0*.*

3.3. Models Arising as Special Cases

Cost register automata and weighted automata. Deterministic VAF with $\top$ as filter class are identical to the *cost register automata* of [1]. The expressiveness results therein relating different restrictions of cost register automata are summed up next:

Theorem 3.5 ([1]) *With* $(K, +, \times)$ *a semiring:*

1. $\mathrm{DetVAF}(K, \mathsf{Scale}^{\mathrm{cl}}, \top)$
   $\subsetneq \mathrm{DetVAF}(K, \top^{\mathrm{cl}}, \top) = \mathrm{DetVAF}(K, \mathsf{Scale}, \top) = \mathrm{UnWA}(K)$
   $\subsetneq \mathrm{DetVAF}(K, \top, \top)$,
2. $\mathrm{DetVAF}(K, \mathsf{Affine}^{\mathrm{cl}}, \top) \subsetneq \mathrm{DetVAF}(K, \mathsf{Affine}, \top) = \mathrm{WA}(K)$.

Moreover, weighted automata are naturally expressed as nondeterministic VAF where only linear transformations are used—in which case, if no register moves are allowed and no filtering is made, having multiple registers does not increase the computing power. Hence, in light of the last point of Theorem 3.5:

Proposition 3.6 *With* $(K, +, \times)$ *a semiring,* $\mathrm{VAF}(K, \mathsf{Scale}^{\mathrm{ml,rl}}, \top) = \mathrm{DetVAF}(K, \mathsf{Affine}, \top)$.

Weighted automata over *valuation monoids* constitute a recent fruitful effort towards generalizing weighted automata to less restricted settings [7]. The direction taken there is to move away from the iterative aspect of weighted automata. Indeed, a valuation monoid M is equipped with an extra function $M^+ \to M$ intended to compute a value given the weights of each of the transitions in a path. Locality and incrementality being at the heart of our models, such automata do not seem to have an equivalent expression within VAF.

Parikh automata and affine Parikh automata. Parikh automata were introduced by Klaedkte and Rueß [10] and further studied and extended in [3]. They are defined as *recognizers*. In a Parikh automaton, a set of (integer) counters is incremented during a run, and a word is accepted if the values belong to a prescribed semilinear set. An *affine Parikh automaton* is defined similarly, except that the update function is lifted to any affine transformation. Viewing languages as functions from Σ^* to $\{0, 1\}$, it is readily seen that:

Proposition 3.7 *Deterministic, unambiguous, and nondeterministic Parikh automata (resp. affine Parikh automata) can be simulated by the corresponding sort of* $\mathrm{VAF}(\mathbb{N}, \mathsf{Trans}, \mathrm{FO}[+])$ *(resp.* $\mathrm{VAF}(\mathbb{N}, \mathsf{Affine}, \mathrm{FO}[+])$*).*

Theorem 3.8 ([3, 4])
- $\mathrm{DetVAF}(\mathbb{N}, \mathsf{Trans}^{\mathrm{ml}}, \mathrm{FO}[+])$ $\subsetneq$ $\mathrm{UnVAF}(\mathbb{N}, \mathsf{Trans}^{\mathrm{ml}}, \mathrm{FO}[+]) = \mathrm{DetVAF}(\mathbb{N}, \mathsf{Trans}, \mathrm{FO}[+]) = \mathrm{UnVAF}(\mathbb{N}, \mathsf{Trans}, \mathrm{FO}[+])$ $\subsetneq$ $\mathrm{OneVAF}(\mathbb{N}, \mathsf{Trans}^{\mathrm{ml}}) \subseteq \mathrm{VAF}(\mathbb{N}, \mathsf{Trans}^{\mathrm{ml}}, \mathrm{FO}[+]) = \mathrm{VAF}(\mathbb{N}, \mathsf{Trans}, \mathrm{FO}[+])$
- $\mathrm{DetVAF}(\mathbb{N}, \mathsf{Affine}, \mathrm{FO}[+]) = \mathrm{UnVAF}(\mathbb{N}, \mathsf{Affine}, \mathrm{FO}[+]) \subsetneq \mathrm{VAF}(\mathbb{N}, \mathsf{Affine}, \mathrm{FO}[+])$*,*
- $\mathrm{VAF}(\mathbb{N}, \mathsf{Trans}, \mathrm{FO}[+])$ *and* $\mathrm{DetVAF}(\mathbb{N}, \mathsf{Affine}, \mathrm{FO}[+])$ *are incomparable.*
- *In these classes, substituting* $\mathbb{N}$ *by* $\mathbb{Z}$ *or* $\mathbb{Q}$ *does not impact the the 0-support languages thus defined.*

Finally, VAFs can keep an extra register to 1 and use the finalization function to return it; if the collapse function of the algebraic structure is the sum, this counts the accepting paths:

Proposition 3.9 *For any VAF, the function mapping a word to the number of paths whose valuations are in the filter is a function in the same class of VAFs. In particular, a VAF can compute the number of accepting paths in a Parikh automaton or an affine Parikh automaton.*

4. Conclusion

In this short exposition, we presented a new model that aims at identifying where some properties of models widely studied in the literature come from. In a longer version, we will present the closure properties, decidability properties, and expressiveness results that hold for the general setting. Our goal is then to identify the essential characteristics that falsify a given property. For instance, unambiguity adds no expressiveness as long as the set of functions $x_i \leftarrow x_j$ is available as updates (as witnessed here by Theorem 3.8). Similarly, some separations between the deterministic and nondeterministic variants can be deduced from the outset, generalizing results for special cases. Another line of results is to work towards simplifying the filter set—for instance, using $\emptyset[+]$ instead of $\mathrm{FO}[+]$ does not impact the expressiveness. Finally, for well-behaved algebraic structures, complexity results can be derived for the decidable problems.

References

[1] R. ALUR, L. D'ANTONI, J. DESHMUKH, M. RAGHOTHAMAN, Y. YUAN, Regular functions and cost register automata. In: *LICS*. IEEE Computer Society, 2013, 13–22. `http://dl.acm.org/citation.cfm?id=2591425`

[2] B. AMINOF, O. KUPFERMAN, R. LAMPERT, Reasoning about online algorithms with weighted automata. *ACM Transactions on Algorithms* 6 (2010) 2, 1–36. `http://portal.acm.org/citation.cfm?doid=1721837.1721844`

[3] M. CADILHAC, A. FINKEL, P. MCKENZIE, Affine Parikh automata. *RAIRO - Theoretical Informatics and Applications* 46 (2012) 04, 511–545.

[4] M. CADILHAC, A. FINKEL, P. MCKENZIE, Unambiguous constrained automata. *Int. J. of Foundations of Computer Science* 24 (2013) 07, 1099–1116.
http://www.worldscientific.com/doi/abs/10.1142/S0129054113400339

[5] K. CHATTERJEE, L. DOYEN, T. A. HENZINGER, Quantitative languages. *ACM Transactions on Computational Logic* 11 (2010) 4, 23.
http://dl.acm.org/citation.cfm?id=1805953

[6] M. DROSTE, W. KUICH, H. VOGLER, *Handbook of Weighted Automata*. 1st edition, Springer Publishing Company, Incorporated, 2009.

[7] M. DROSTE, I. MEINECKE, Weighted automata and regular expressions over valuation monoids. *Int. J. of Foundations of Computer Science* 22 (2011) 08, 1829–1844.
http://www.worldscientific.com/doi/abs/10.1142/S0129054111009069

[8] S. GINSBURG, E. H. SPANIER, Semigroups, Presburger formulas and languages. *Pacific J. Mathematics* 16 (1966) 2, 285–296.

[9] O. H. IBARRA, Reversal-Bounded Multicounter Machines and Their Decision Problems. *J. ACM* 25 (1978) 1, 116–133.

[10] F. KLAEDTKE, H. RUESS, Monadic Second-Order Logics with Cardinalities. In: *ICALP*. LNCS 2719, Springer-Verlag, 2003, 681–696.

[11] V. MITRANA, R. STIEBE, Extended finite automata over groups. *Discrete Appl. Math.* 108 (2001) 3, 287–300.

[12] J. SAKAROVITCH, *Elements of Automata Theory*. Cambridge University Press, 2009.

PROPERTIES OF INPUT-DRIVEN QUEUE AUTOMATA WITH INTERNAL TRANSDUCTIONS

Martin Kutrib Andreas Malcher
Matthias Wendlandt

Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
{kutrib,malcher,matthias.wendlandt}@informatik.uni-giessen.de

Abstract
For input-driven queue automata (IDQA) the input alphabet is divided into three distinct classes and the actions on the queue (enter, remove, nothing) are solely governed by the input symbols. Here, this model is extended in such a way that the input of an IDQA is preprocessed by an internal deterministic sequential transducer. These automata are called tinput-driven queue automata (TDQA), and it turns out that TDQAs are more powerful than IDQAs. The work in progress paper studies the properties of tinput-driven queue automata with weak, that is, deterministic injective and length-preserving, internal transducers.

1. Introduction

Finite automata possess many nice properties such as equivalence of nondeterministic and deterministic models, existence of minimization algorithms, closure under many operations, and decidable questions such as emptiness, inclusion, or equivalence. On the other hand, their computational power is quite low since only regular languages are accepted. It is therefore natural to consider extensions of the model featuring additional storage media such as pushdown stores, stacks, or queues. In general, such extensions lead to a broader family of accepted languages, but also to a weaker manageability of the models since certain closure properties do not longer hold, minimization algorithms do not exist, and formerly decidable questions become undecidable. Thus, there is an obvious interest in extensions which enlarge the language family, but keep as many of the 'good' properties as possible.

One such extension is represented by input-driven automata. Basically, for such devices the operations on the storage medium are dictated by the input symbols. The first references date back to [13, 11], where input-driven pushdown automata are introduced in which the input symbols define whether a push operation, a pop operation, or no operation on the pushdown store has to be performed. A recent survey with many valuable references on complexity

aspects of input-driven pushdown automata may be found in [12]. Extensions of the model with respect to multiple pushdown stores or more general auxiliary storages are introduced in [9, 10]. Recently, the computational power of input-driven automata using the storage medium of a stack and a queue, respectively, have been investigated in [3, 8]. Here we are particularly interested in *queue automata*. Some contributions relate these devices to other well-known concepts in formal language theory and theoretical computer science. For instance, in [6] queue automata (there called Post machines) with certain features are shown to characterize the class of languages accepted by multi-reset machines [4], as well as some classes of languages defined by equality sets. In [7], a restricted version of context-free grammars, called breadth-first grammars, are provided as a generating system for certain classes of languages accepted by queue automata.

The edge between languages that are accepted by input-driven queue automata (IDQA) or not is very small. For example, language $\{ a^n\$b^n \mid n \geq 1 \}$ is accepted by an IDQA where an a means an enter operation, b means a remove operation, and a $\$$ leaves the queue unchanged. On the other hand, the very similar language $\{ a^n\$a^n \mid n \geq 1 \}$ is not accepted by any IDQA. Similarly, the language $\{ w\$w \mid w \in \{a,b\}^+ \}$ is not accepted by any IDQA, but if the second w is written down with some marked alphabet $\{\hat{a},\hat{b}\}$, then language $\{ w\$\hat{w} \mid w \in \{a,b\}^+ \}$ is accepted by an IDQA. To overcome these obstacles we provide the input-driven queue automaton with an internal sequential transducer that preprocesses the input. In the first example above such a transducer translates every a before reading $\$$ to a and after reading $\$$ to b. An IDQA with internal transducer is said to be *tinput-driven* (TDQA). While an internal transducer does not affect the computational capacity of general queue automata, it clearly does for *input-driven* versions. To implement the idea without giving the transducers too much power for the overall computation, essentially, we will consider only deterministic injective and length-preserving transducers.

2. Preliminaries

Let Σ^* denote the set of all words over the finite alphabet Σ. The *empty word* is denoted by λ, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The set of words of length at most $n \geq 0$ is denoted by $\Sigma^{\leq n}$. The *reversal* of a word w is denoted by w^R. For the *length* of w we write $|w|$. We use $\subseteq$ for *inclusions* and $\subset$ for *strict inclusions*.

A classical deterministic queue automaton is called input-driven if the next input symbol defines the next action on the queue, that is, entering a symbol at the end of the queue, removing a symbol from the front of the queue, or changing the internal state without modifying the queue content. To this end, we assume that the input alphabet Σ is partitioned into the sets Σ_D, Σ_R, and Σ_N, that control the actions enter (D), remove (R), and state change only (N). Such a partition is called a *signature*. A formal definition is:

Definition 2.1 *A* deterministic input-driven queue automaton*, abbreviated as IDQA, is a system* $M = \langle Q, \Sigma, \Gamma, q_0, F, \bot, \delta_D, \delta_R, \delta_N \rangle$*, where*

1. Q *is the finite set of* internal states,
2. Σ *is the finite set of* input symbols *consisting of the disjoint union of sets* Σ_D, Σ_R, *and* Σ_N,
3. Γ *is the finite set of* queue symbols,
4. $q_0 \in Q$ *is the* initial state,
5. $F \subseteq Q$ *is the set of* accepting states,
6. $\perp \notin \Gamma$ *is the* empty-queue symbol,
7. δ_D *is the partial transition function mapping* $Q \times \Sigma_D \times (\Gamma \cup \{\perp\})$ *to* $Q \times \Gamma$,
8. δ_R *is the partial transition function mapping* $Q \times \Sigma_R \times (\Gamma \cup \{\perp\})$ *to* Q,
9. δ_N *is the partial transition function mapping* $Q \times \Sigma_N \times (\Gamma \cup \{\perp\})$ *to* Q.

A *configuration* of an IDQA $M = \langle Q, \Sigma, \Gamma, q_0, F, \perp, \delta_D, \delta_R, \delta_N \rangle$ is a triple (q, w, s), where $q \in Q$ is the current state, $w \in \Sigma^*$ is the unread part of the input, and $s \in \Gamma^*$ denotes the current queue content, where the leftmost symbol is at the front. Thus, the *initial configuration* for an input string w is set to (q_0, w, λ). During the course of its computation, M runs through a sequence of configurations. One step from a configuration to its successor configuration is denoted by $\vdash$. Let $a \in \Sigma$, $w \in \Sigma^*$, $z, z' \in \Gamma$, and $s \in \Gamma^*$. We set

1. $(q, aw, zs) \vdash (q', w, zsz')$, if $a \in \Sigma_D$ and $(q', z') = \delta_D(q, a, z)$,
2. $(q, aw, \lambda) \vdash (q', w, z')$, if $a \in \Sigma_D$ and $(q', z') = \delta_D(q, a, \perp)$,
3. $(q, aw, zs) \vdash (q', w, s)$, if $a \in \Sigma_R$ and $q' = \delta_R(q, a, z)$,
4. $(q, aw, \lambda) \vdash (q', w, \lambda)$, if $a \in \Sigma_R$ and $q' = \delta_R(q, a, \perp)$,
5. $(q, aw, zs) \vdash (q', w, zs)$, if $a \in \Sigma_N$ and $q' = \delta_N(q, a, z)$,
6. $(q, aw, \lambda) \vdash (q', w, \lambda)$, if $a \in \Sigma_N$ and $q' = \delta_N(q, a, \perp)$.

So, whenever the queue is empty, the successor configuration is computed according to the definition of the transition functions on the special empty-queue symbol $\perp$. We denote the reflexive and transitive (respectively, transitive) closure of $\vdash$ by $\vdash^*$ (respectively, $\vdash^+$). The language accepted by the IDQA M is the set $L(M)$ of words for which there exists a computation beginning in the initial configuration and ending in a configuration in which the whole input is read and an accepting state is entered. Formally:

$$L(M) = \{\, w \in \Sigma^* \mid (q_0, w, \lambda) \vdash^* (q, \lambda, s) \text{ with } q \in F, s \in \Gamma^* \,\}.$$

For the definition of tinput-driven queue automata we need the notion of *deterministic one-way sequential transducers* (DST) which are basically deterministic finite automata equipped with an initially empty output tape. In every transition a DST appends a string over the output alphabet to the output tape. The transduction defined by a DST is the set of all pairs (w, v), where w is the input and v is the output produced after having read w completely. Formally, a DST is a system $T = \langle Q, \Sigma, \Delta, q_0, \delta \rangle$, where Q is the finite set of internal states, Σ is the finite set of input symbols, Δ is the finite set of output symbols, $q_0 \in Q$ is the initial state, and δ is the partial transition function mapping $Q \times \Sigma$ to $Q \times \Delta^*$. By $T(w) \in \Delta^*$ we denote the output produced by T on input $w \in \Sigma^*$. Here we will consider only injective and length-preserving DSTs which are also known as injective Mealy machines.

Let M be an IDQA and T be an injective and length-preserving DST so that the output alphabet of T is the input alphabet of M. The pair (M,T) is called a *tinput-driven queue automaton* (TDQA) and the language accepted by (M,T) is $L(M,T) = \{ w \in \Sigma^* \mid T(w) \in L(M) \}$.

For a computation of a queue automaton, a turn is a phase in which the length of the queue first increases and then decreases. Formally, a sequence of at least three configurations $(q_1, w_1, s_1) \vdash (q_2, w_2, s_2) \vdash \cdots \vdash (q_m, w_m, s_m)$ is a *turn* if $|s_1| < |s_2| = \cdots = |s_{m-1}| > |s_m|$. For any given $k \geq 0$, a *k-turn* computation is any computation containing exactly k turns.

A TDQA performing *at most* k turns in *any* computation is called k-turn TDQA and will be denoted by TDQA_k. Analogously, k-turn IDQA are defined, and will be denoted by IDQA_k.

In order to clarify this notion we continue with an example.

Example 2.2 *Language $L_1 = \{ a^n \$ a^n \mid n \geq 1 \}$ is accepted by a TDQA. Before reading symbol $\$$ the transducer maps an a to an a, and after reading $\$$ it maps an a to a b. Thus, L_1 is translated to $\{ a^n \$ b^n \mid n \geq 1 \}$ which is accepted by some IDQA.*

Similarly, $L_2 = \{ w \$ w \mid w \in \{a,b\}^* \}$ can be accepted by some TDQA. Here, the transducer maps any a, b to a, b before reading $\$$ and to $\hat{a}, \hat{b}$ after reading $\$$. This gives the language $\{ w \$ \hat{w} \mid w \in \{a,b\}^* \}$ which clearly is accepted by some IDQA.

Finally, consider $L_3 = \{ a^n b^{2n} \mid n \geq 1 \}$. Here, the transducer maps an a to a and every b alternately to b and c. This gives language $\{ a^n (bc)^n \mid n \geq 1 \}$ which is accepted by some IDQA: every a implies an enter-operation, every b implies a remove, and every c leaves the queue unchanged. □

3. Determinization

Each language accepted by a nondeterministic input-driven pushdown automaton is also accepted by a deterministic input-driven pushdown automaton. The simulation costs for an n-state nondeterministic input-driven pushdown automaton are $2^{\Theta(n^2)}$ states [13]. It is shown in [2] that this size is necessary in the worst case.

The basic idea of the proof is that the automaton stores applicable transition rules onto the pushdown store, when a push operation should be done, instead of pushing the appropriate symbol. The actual push operation is simulated at the time at which the symbol to be pushed is popped. In this way, it may happen that some transition rule pushed does not belong to a valid computation. However, the construction allows to distinguish these cases and, thus, only valid transitions will be evaluated.

This technique can not be assigned to input-driven queue automata. The reason is that on input-driven pushdown automata the last symbol pushed is used first. Thus it can be determined whether it belongs to a valid computation or not. In contrast, input-driven queue automata work according to the FIFO principle. Therefore, there could be many symbols in

between the queue symbol currently to be removed and the symbol that has been entered last. So, the technique does not allow to verify that the remove operation simulates only enter operations belonging to a valid computation.

4. Closure Properties

We investigate the closure properties of TDQAs with a possibly unbounded number of turns and with a fixed number k of turns. Let us start with a summary of the results for the former class of TDQAs. A first result is that TDQAs are closed under complementation. For the remaining Boolean operations of union and intersection we have to distinguish whether or not the given two TDQAs in question have identical or at least compatible signatures. Clearly, two TDQAs M and M' have identical signatures if they are defined over the same input alphabet and the behavior of the queue of M and M' is identical for all input symbols. In case of *compatible* signatures, the input alphabets of M and M' may differ, but the behavior of the queue of M and M' is identical for all input symbols belonging to the intersection of both input alphabets. For two TDQAs with compatible signatures it is possible to construct TDQAs for the union and the intersection. It should be noted that we obtain as a by-product of the latter construction that TDQAs are closed under intersection with regular languages. In contrast, TDQAs are *not* closed under union and intersection in case of incompatible signatures.

Next, we investigate the operations of concatenation and Kleene star. For concatenation we distinguish between the concatenation of two TDQAs with compatible and not necessarily compatible signatures. It turns out that TDQAs are *not* closed under concatenation in either case. It is not difficult to extend this non-closure result to the non-closure of TDQAs under Kleene star.

Finally, we investigate a possible closure under homomorphic replacements. We obtain that TDQAs are *not* closed even under length-preserving homomorphisms which are a weak form of homomorphic replacement. On the other hand, we expect the closure under inverse homomorphism which is in line with a similar result for tinput-driven pushdown automata. On the other hand, it should be noted that input-driven queue automata are not closed under inverse homomorphism [8].

For turn-bounded TDQAs we yield similar results. First, we observe that all non-closure results obtained for TDQAs carry over to the turn-bounded case. Additionally, it is possible to show that TDQA$_k$s are *not* closed under complementation. On the other hand, we still have the closure under union and intersection in case of compatible signatures and under inverse homomorphism.

5. Decidability Questions

TDQAs with an unbounded number of turns are a powerful model, since they can encode in a suitable way the computations of linear bounded automata. Owing to this encoding it is possible to reduce decidability questions for linear bounded automata to decidability questions for TDQAs. Since it is known that for the latter class of automata many decidability questions are undecidable and not even semidecidable, we obtain by our reduction that the commonly studied questions of emptiness, finiteness, inclusion, equivalence, regularity, and context-freeness are all undecidable and not semidecidable for TDQAs.

In the turn-bounded case, it is possible to show the decidability of emptiness and finiteness for TDQA_ks. Both results are mainly based on the fact that the language accepted by a TDQA_k can be shown to be letter-equivalent to a context-free language. Furthermore, the questions of equivalence and inclusion are decidable for TDQA_ks under the condition that both TDQAs in question have compatible signatures. On the other hand, in case of incompatible signatures the inclusion problem turns again into an undecidable and not semidecidable problem. It is currently an open question whether the equivalence of TDQAs becomes undecidable in case of incompatible signatures. For TDQAs with an unbounded number of turns, we obtain that the questions of inclusion and equivalence remain undecidable and not semidecidable even if the TDQAs are provided with compatible signatures.

References

[1] R. ALUR, P. MADHUSUDAN, Visibly pushdown languages. In: L. BABAI (ed.), *Symposium on Theory of Computing (STOC 2004)*. ACM, 2004, 202–211.

[2] R. ALUR, P. MADHUSUDAN, Adding nesting structure to words. *J. ACM* 56 (2009).

[3] S. BENSCH, M. HOLZER, M. KUTRIB, A. MALCHER, Input-Driven Stack Automata. In: J. C. M. BAETEN, T. BALL, F. S. DE BOER (eds.), *Theoretical Computer Science (TCS 2012)*. LNCS 7604, Springer, 2012, 28–42.

[4] R. V. BOOK, S. A. GREIBACH, C. WRATHALL, Reset Machines. *J. Comput. System Sci.* 19 (1979), 256–276.

[5] H. BORDIHN, M. HOLZER, M. KUTRIB, Economy of Description for Basic Constructions on Rational Transductions. *J. Autom., Lang. Comb.* 9 (2004), 175–188.

[6] F. BRANDENBURG, Multiple Equality Sets and Post Machines. *J. Comput. System Sci.* 21 (1980), 292–316.

[7] A. CHERUBINI, C. CITRINI, S. CRESPI-REGHIZZI, D. MANDRIOLI, QRT FIFO Automata, Breadth-First Grammars and Their Relations. *Theoret. Comput. Sci.* 85 (1991), 171–203.

[8] M. KUTRIB, A. MALCHER, C. MEREGHETTI, B. PALANO, M. WENDLANDT, Input-Driven Queue Automata: Finite Turns, Decidability, and Closure Properties. *Theoret. Comput. Sci.* 578 (2015), 58–71.

[9] S. LA TORRE, P. MADHUSUDAN, G. PARLATO, A Robust Class of Context-Sensitive Languages. In: *Logic in Computer Science (LICS 2007)*. IEEE Computer Society, 2007, 161–170.

[10] P. MADHUSUDAN, G. PARLATO, The tree width of auxiliary storage. In: T. BALL, M. SAGIV (eds.), *Principles of Programming Languages, (POPL 2011)*. ACM, 2011, 283–294.

[11] K. MEHLHORN, Pebbling Moutain Ranges and its Application of DCFL-Recognition. In: J. W. DE BAKKER, J. VAN LEEUWEN (eds.), *International Colloquium on Automata, Languages and Programming (ICALP 1980)*. LNCS 85, Springer, 1980, 422–435.

[12] A. OKHOTIN, K. SALOMAA, Complexity of input-driven pushdown automata. *SIGACT News* 45 (2014), 47–67.

[13] B. VON BRAUNMÜHL, R. VERBEEK, Input-Driven Languages are Recognized in $\log n$ Space. In: M. KARPINSKI, J. VAN LEEUWEN (eds.), *Topics in the Theory of Computation*. Mathematics Studies 102, North-Holland, Amsterdam, 1985, 1–19.

THE TREE COMPLETION OF AN AUTOMATON GROUP

Francesco Matucci[(A)] Pedro V. Silva[(B)]

[(A)]Instituto de Matemática, Estatística e Computação Científica, Universidade Estadual de Campinas, Rua Sérgio Buarque de Holanda, 651
Cidade Universitária "Zeferino Vaz" 13083-859, Barão Geraldo, Campinas, São Paulo, Brazil
francesco@ime.unicamp.br

[(B)]Centro de Matemática, Faculdade de Ciências, Universidade do Porto,
R. Campo Alegre 687, 4169-007 Porto, Portugal
pvsilva@fc.up.pt

Abstract
If d denotes the depth metric on the regular A-ary rooted tree T_A, then (Aut, d_A) is a compact complete metric space. Each automaton group G on A determines a compact complete subgroup $\overline{G}$ of (Aut, d_A), named the tree completion of G. Several natural problems arise in connection with $\mathrm{Aut}(G)$ and $\overline{G}$. We report on results obtained for the subclass of automata groups defined by the Cayley machine of a finite abelian group H, including the famous lamplighter group.

1. Introduction

Given a finite nonempty set A, we may identify the free monoid A^* with the regular A-ary rooted tree T_A, where A^* is the set of nodes, the empty word ε is the root and ua is a son of u for all $u \in A^*$ and $a \in A$. An *automorphism* of T_A is then a permutation of A^* which preserves length and the prefix relation. Note that $\mathrm{Aut}(T_A)$ is a subgroup of the symmetric group on A^*.

The group $\mathrm{Aut}(T_A)$ is canonically isomorphic to the infinite wreath product $S_A \wr S_A \wr \ldots$, where S_A denotes the symmetric group on A. This infinite decomposition involves the so-called local permutations, which we now define.

Given $\varphi \in \mathrm{Aut}(T_A)$ and $u \in A^*$, there exists some $\varphi_u \in S_A$ such that

$$(ua)\varphi = (u\varphi)(a\varphi_u) \text{ for every } a \in A.$$

[(A)]The first author was partially supported by CAUL, project PEst-OE/MAT/UI0143/2014 of FCT and UID/Multi/04621/2013 of CEMAT.

[(B)]The second author was partially supported by CMUP (UID/MAT/00144/2013), which is funded by FCT (Portugal) with national (MEC) and European structural funds through the programs FEDER, under the partnership agreement PT2020.

We say that φ_u is the local permutation induced by φ at vertex u.

We define also the *cone automorphism* $\varphi_{uA^*} \in \mathrm{Aut}(T_A)$ through

$$(uv)\varphi = (u\varphi)(v\varphi_{uA^*}) \quad (v \in A^*).$$

We say that $G \leq \mathrm{Aut}(T_A)$ is a *self-similar* group if

$$\forall \varphi \in G \; \forall u \in A^* \quad \varphi_{uA^*} \in G.$$

Finitely generated self-similar groups can be constructed with the help of finite automata/transducers of a particular type.

Indeed, a *Mealy machine* is a structure of the form $\mathcal{M} = (A, Q, \delta, \lambda)$, where:

- A is a finite nonempty set (alphabet);
- Q is a finite nonempty set (state set);
- $\delta : Q \times A \to Q$ is a function (transition function);
- $\lambda : Q \times A \to A$ is a function (output function).

We say that $\mathcal{M}$ is *invertible* if the mapping

$$\begin{aligned} \lambda_q : A &\to A \\ a &\mapsto (q, a)\lambda \end{aligned}$$

is a permutation for every $q \in Q$.

We define recursively an action $Q \times A^* \to Q$ by

- $q\varepsilon = q$;
- $q(ua) = (qu, a)\delta \quad (u \in A^*,\ a \in A)$.

For every $q \in Q$, we extend λ_q to a mapping $\widehat{\lambda}_q : A^* \to A^*$ by setting

- $\varepsilon\widehat{\lambda}_q = \varepsilon$;
- $(ua)\widehat{\lambda}_q = (u\widehat{\lambda}_q)(a\lambda_{qu}) \quad (u \in A^*,\ a \in A)$.

If $\mathcal{M}$ is invertible then $\widehat{\lambda}_q$ is a permutation of A^*, indeed $\widehat{\lambda}_q \in \mathrm{Aut}(T_A)$. The *automaton group* generated by $\mathcal{M}$ is the (finitely generated) subgroup of $\mathrm{Aut}(T_A)$ generated by $\{\widehat{\lambda}_q \mid q \in Q\}$. It will be denoted by $\mathcal{G}(\mathcal{M})$. Automata groups are precisely the finitely generated self-similar groups.

Let A be a finite nonempty alphabet. We define a metric on $\mathrm{Aut}(T_A)$ as follows. Given $\varphi, \psi \in \mathrm{Aut}(T_A)$, let

$$d(\varphi, \psi) = \begin{cases} 2^{-\min\{n \in \mathbb{N} \,\mid\, \varphi|_{A^n} \neq \psi|_{A^n}\}} & \text{if } \varphi \neq \psi \\ 0 & \text{if } \varphi = \psi \end{cases}$$

where A^n denote the n-th level of the tree T_A. It is immediate that d is indeed an ultrametric on $\mathrm{Aut}(T_A)$, which we call the *depth metric*. As it is remarked in [1], the metric space $(\mathrm{Aut}(T_A), d)$ is compact and therefore complete.

Let $G \leq \mathrm{Aut}(T_A)$. We denote by $\overline{G}$ the topological closure of G in $(\mathrm{Aut}(T_A), d)$. Note that $\overline{G}$, being a closed subset of a compact space, is itself compact (and therefore complete). The restrictions of the depth metric to either G or $\overline{G}$ will still be denoted by d and referred to as depth metric.

It is easy to check that $\overline{G}$ consists precisely of those $\varphi \in \mathrm{Aut}(T_A)$ such that

$$\forall n \in \mathbb{N}\ \exists \varphi_n \in G : \varphi|_{A^n} = \varphi_n|_{A^n},$$

and is indeed the completion of (G, d). It has been introduced for branch groups by Bartholdi, Grigorchuk and Šunić (see [1, Definition 1.18]) under the name *tree completion*, which we also adopt. Note that $\overline{G}$ is a subgroup of $\mathrm{Aut}(T_A)$.

The topology on G induced by d is none other than the *topology of pointwise convergence*: if we consider A^* endowed with the discrete topology, then $\lim_{n\to+\infty} \varphi_n = \varphi$ holds in G if and only if

$$\forall u \in A^* \quad \lim_{n\to+\infty} u\varphi_n = u\varphi,$$

i.e. each sequence $(u\varphi_n)_n$ is stationary with limit $u\varphi$.

Several natural problems arise in connection with $\mathrm{Aut}(G)$ and $\overline{G}$:

(P1) Does every automorphism of G admit a continuous extension $\overline{\varphi}$ to $\overline{G}$?

(P2) Does every automorphism of G admit a (continuous) extension to an automorphism of $\mathrm{Aut}(T_A)$?

(P3) Is every fixed point of $\overline{\varphi}$ a limit point of fixed points of φ?

We shall report on results obtained for the subclass of automata groups defined by the *Cayley machine* of a finite abelian group H (i.e. the wreath product $H \wr \mathbb{Z}$), including the famous *lamplighter group*.

2. Results

2.1. $\overline{G}$ as a Subgroup of $\mathrm{Aut}(T_A)$

Lemma 2.1 *Let $G \leq \mathrm{Aut}(T_A)$ be self-similar. Then $\overline{G}$ is also a self-similar subgroup of $\mathrm{Aut}(T_A)$.*

Automata groups, being finitely generated, are always countable. However, the next result shows that their tree completions are countable only in trivial cases.

Proposition 2.2 *Let $G \leq \mathrm{Aut}(T_A)$.*

(i) If G is finite, then $\overline{G} = G$.
(ii) If G is infinite, then $\overline{G}$ is uncountable.

We can prove that $\overline{G} = \mathrm{Aut}(T_A)$ for an automaton group G only in trivial cases. The key is the following lemma, where the *rank* of a finitely generated group G denotes the minimum cardinality of a generating set of G.

We shall denote by $T_A^{(n)}$ the (rooted) subtree of T_A induced by the nodes $A^{\leq n}$.

Lemma 2.3 *Let A be a finite alphabet with $|A| \geq 2$ and let $n \geq 1$. Then* $\mathrm{rank}(\mathrm{Aut}(T_A^{(n)})) \geq n$.

Proposition 2.4 *Let A be a finite alphabet with $|A| \geq 2$ and let $G \leq \mathrm{Aut}(T_A)$ be finitely generated. Then $\overline{G} < \mathrm{Aut}(T_A)$.*

In particular, we have $\overline{G} < \mathrm{Aut}(T_A)$ for every automaton group over an alphabet with at least two letters.

2.2. Automorphisms of an Automaton Group

It is a natural question to enquire which endomorphisms θ of an automaton group admit a continuous extension to $(\overline{G}, d)$. By general topology, this is equivalent to say that θ is uniformly continuous in (G, d). Note that the continuous extension must be unique, and it is easy to see that is also an automorphism if *theta* and θ^{-1} are both uniformly continuous automorphisms.

Given $n \in \mathbb{N}$, we define the *n-th level stabilizer* of $G \leq \mathrm{Aut}(T_A)$ to be

$$\mathrm{Stab}_n(G) = \{\varphi \in G \mid \varphi|_{A^n} = id\}.$$

We can relate uniform continuity to stabilizers in the case of automorphisms.

Proposition 2.5 *Let $G \leq \mathrm{Aut}(T_A)$ and let $\theta : G \to G$ be an automorphism. Then the following conditions are equivalent:*

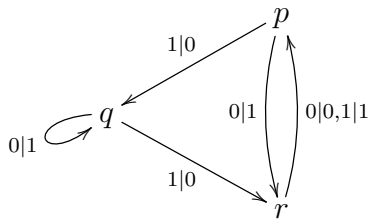
(i) θ is uniformly continuous in (G, d);
(ii) $\forall m \in \mathbb{N}\ \exists n \in \mathbb{N} : (\mathrm{Stab}_n(G))\theta \subseteq \mathrm{Stab}_m(G)$.

Corollary 2.6 *Let $G \leq \mathrm{Aut}(T_A)$ and let $\theta : G \to G$ be an inner automorphism. Then θ is uniformly continuous in (G, d).*

Corollary 2.7 *Let $G \leq \mathrm{Aut}(T_A)$ be such that $\mathrm{Stab}_n(G)$ is fully invariant for every $n \in \mathbb{N}$. Then every automorphism of G is uniformly continuous in (G, d).*

The following example shows that this condition does not hold for every automaton group.

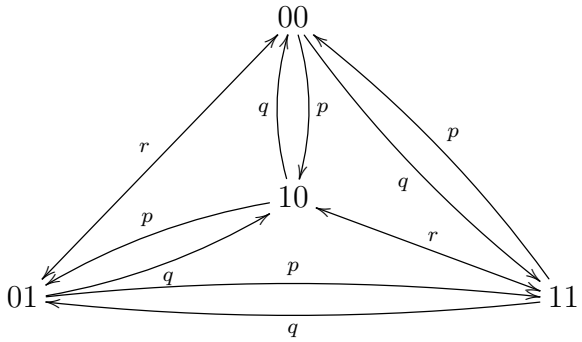
Example 2.8 *Let G be the automaton group generated by Aleshin's automaton*



Then $\mathrm{Stab}_2(G)$ *is not fully invariant. Indeed, Vorobets and Vorobets proved in [4] that G is the free group on $\{\widehat{\lambda}_p, \widehat{\lambda}_q, \widehat{\lambda}_r\}$, hence*

$$\widehat{\lambda}_p \mapsto \widehat{\lambda}_r \widehat{\lambda}_p, \quad \widehat{\lambda}_q \mapsto \widehat{\lambda}_q, \quad \widehat{\lambda}_r \mapsto \widehat{\lambda}_r$$

defines an automorphism θ of G. The action of the generators of G at depth 2 is described by the diagram



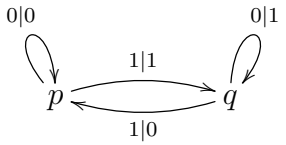
It follows easily that $\widehat{\lambda}_p\widehat{\lambda}_q \in Stab_2(G)$, but $(\widehat{\lambda}_p\widehat{\lambda}_q)\theta = \widehat{\lambda}_r\widehat{\lambda}_p\widehat{\lambda}_q \notin Stab_2(G)$. Therefore $Stab_2(G)$ is not fully invariant.

2.3. Cayley Machines of Finite Abelian Groups

Let H be a finite group. The Cayley machine of H, introduced by Krohn and Rhodes in [2], is the invertible Mealy machine $\mathcal{C}_H = (H, H, \delta, \lambda)$ defined by

$$(h, h')\delta = (h, h')\lambda = hh' \in H \quad (h, h' \in H).$$

If H is abelian, Steinberg and the second author proved in [3] that $\mathcal{G}(\mathcal{C}_H) \cong H \wr \mathbb{Z}$, the wreath product of H and $\mathbb{Z}$. If $G = C_2$ is the group of order 2, we get the Cayley machine



and the *lamplighter group* $\mathcal{L}$.

The group $\mathcal{L}$ is generated by the automorphisms ξ, α of T_2 defined by

$$(x_0, x_1, x_2, \ldots)\xi = (x_0, x_0 + x_1, x_0 + x_1 + x_2, \ldots),$$
$$(x_0, x_1, x_2, \ldots)\alpha = (1 + x_0, x_1, x_2, \ldots).$$

The lamplighter group admits the group presentation

$$\langle \alpha, \xi \mid \alpha^2 = 1, (\xi^m \alpha \xi^{-m})(\xi^n \alpha \xi^{-n}) = (\xi^n \alpha \xi^{-n})(\xi^m \alpha \xi^{-m}) \ (m, n \in \mathbb{Z}) \rangle.$$

Note that $\xi^{m_0}\alpha^{n_1}\xi^{m_1} \ldots \alpha^{n_k}\xi^{m_k}$ has finite order if and only if $m_0 + m_1 + \ldots + m_k = 0$. Let $\mathrm{Fin}(\mathcal{L})$ denote the subset of all elements of $\mathcal{L}$ of finite order. Then $\mathrm{Fin}(\mathcal{L})$ is an abelian normal subgroup of $\mathcal{L}$ and $\mathcal{L}/\mathrm{Fin}(\mathcal{L})$ is infinite cyclic. Moreover, $\mathrm{Fin}(\mathcal{L})$ is a direct sum of countably many groups of order 2.

We describe next the automorphisms of $\mathcal{L}$. With that purpose, we introduce the following notation.

For every $m \in \mathbb{Z}$, let $\beta_m = \xi^m \alpha \xi^{-m}$.

Proposition 2.9 *The automorphisms of $\mathcal{L}$ are the the endomorphisms $\varphi_{n_1,\ldots,n_k,\delta,m}$ defined by*

$$\xi\varphi_{n_1,\ldots,n_k,\delta,m} = \beta_{n_1} \ldots \beta_{n_k}\xi^{\delta}, \quad \alpha\varphi_{n_1,\ldots,n_k,\delta,m} = \beta_m,$$

where $k \geq 0$; $n_i, m \in \mathbb{Z}$; $n_1 < \ldots < n_k$; $\delta = \pm 1$.

Let $\mathrm{Aut}^+(\mathcal{L})$ denote the set of all automorphism of the form $\varphi_{n_1,\ldots,n_k,1,m}$ (positive automorphisms). Then $\mathrm{Aut}^+(\mathcal{L})$ is a (normal) subgroup of index 2 of $\mathrm{Aut}(\mathcal{L})$. We denote by $\mathrm{Inn}(\mathcal{L})$ the subgroup of inner automorphisms of $\mathcal{L}$. Note that $\mathrm{Inn}(\mathcal{L}) \leq \mathrm{Aut}^+(\mathcal{L})$.

Proposition 2.10 *(i)* $\mathrm{Aut}^+(\mathcal{L}) \cong \mathcal{L}$;
(ii) $\mathrm{Aut}(\mathcal{L}) \cong \mathcal{L} \rtimes C_2$.

Lemma 2.11 *(i)* $\mathrm{Aut}^+(\mathcal{L}) = \langle \varphi_{1,1}, \varphi_{0,1,0} \rangle$;
(ii) $\mathrm{Aut}(\mathcal{L}) = \langle \varphi_{1,1}, \varphi_{0,1,0}, \varphi_{-1,0} \rangle$.

Lemma 2.12 *Let $m \geq 0$.*

(i) Let $n \in \mathbb{Z}$. Then $\xi^n \in \mathrm{Stab}_{2^m}(\mathcal{L})$ if and only if $2^{m+1} | n$.
(ii) Let $i_1, \ldots, i_r \in \mathbb{Z}$. Then $\beta_{i_1} \ldots \beta_{i_r} \in \mathrm{Stab}_{2^m-1}(\mathcal{L})$ if and only if the numbers

$$|\{j \in \{1, \ldots, r\} \mid i_j \equiv p \,(\mathrm{mod}\, 2^m)\}|$$

have the same parity for $p = 0, 1, \ldots, 2^m - 1$.

Theorem 2.13 *(i) Every automorphism of $\mathcal{L}$ is uniformly continuous for the depth metric.*
(ii) Every positive automorphism of $\mathcal{L}$ admits a continuous extension to an automorphism of $\mathrm{Aut}(T_A)$.

The following example shows that the fixed point subgroup of an automorphism of $\mathcal{L}$ needs not be finitely generated, even if the automorphism is inner.

Example 2.14 *There exists* $\theta \in \mathrm{Inn}(\mathcal{L})$ *such that* $\mathrm{Fix}(\theta)$ *is not finitely generated.*

Indeed, let $\theta \in \mathrm{Inn}(\mathcal{L})$ be defined by

$$\psi\theta = \alpha\psi\alpha^{-1} \quad (\psi \in \mathcal{L}).$$

Since $\mathrm{Fin}(\mathcal{L})$ is abelian, we have

$$\gamma\theta = \alpha\gamma\alpha = \alpha^2\gamma = \gamma$$

for every $\gamma \in \mathrm{Fin}(\mathcal{L})$. On the other hand, given $n \in \mathbb{Z}$, we have

$$\xi^n\theta = \alpha\xi^n\alpha = \beta_0\beta_n\xi^n,$$

hence $\xi^n \in \mathrm{Fix}(\theta)$ if and only if $n = 0$.

It follows that $\mathrm{Fix}(\theta) = \mathrm{Fin}(\mathcal{L})$, an infinite abelian torsion group, therefore non finitely generated.

3. Conclusions

The results obtained so far, which seem to be generalizable beyond the case of the lamplighter group, encourage us to conjecture that problems (P1) and (P2) may admit a positive solution. We have made no progress yet on problem (P3).

References

[1] L. BARTHOLDI, R. I. GRIGORCHUK, Z. ŠUNIĆ, Branch groups. In: *Handbook of Algebra, Vol. 3*. North-Holland, Amsterdam, 2003, 989–1112.

[2] K. KROHN, J. RHODES, Algebraic theory of machines. In: *Proc. Sympos. Math. Theory of Automata (New York, 1962)*. Polytechnic Press of Polytechnic Inst. of Brooklyn, Brooklyn, N.Y.,, 1963.

[3] P. V. SILVA, B. STEINBERG, On a class of automata groups generalizing lamplighter groups. *Internat. J. Algebra Comput.* 15 (2005) 5, 1213–1234.

[4] M. VOROBETS, Y. VOROBETS, On a free group of transformations defined by an automaton. *Geom. Dedicata* 124 (2007), 237–249.

Author Index